# Strict General Setting for Building Decision Procedures into Theorem Provers


**Division of Informatics, University of Edinburgh**

**Centre for Intelligent Systems and their Applications**

# Strict General Setting for Building Decision Procedures into Theorem Provers

by

Predrag Janicic, Alan Bundy

**Informatics Research Report EDI-INF-RR-0097**

**Division of Informatics** **January 2001**
http://www.informatics.ed.ac.uk/

# Strict General Setting for Building Decision Procedures into Theorem Provers

Predrag Janicic, Alan Bundy

Informatics Research Report EDI-INF-RR-0097

DIVISION *of* INFORMATICS
Centre for Intelligent Systems and their Applications

January 2001

appears in Procs of ISCAR 2001

**Abstract :**

The efficient and flexible incorporating of decision procedures into theorem provers is very important for their successful use. There are several approaches for combining and augmenting of decision procedures; some of them support handling uninterpreted functions, congruence closure, lemma invoking etc. In this paper we present a variant of one general setting for building decision procedures into theorem provers (gs framework [18]). That setting is based on macro inference rules motivated by techniques used in different approaches. The general setting enables a simple describing of different combination/augmentation schemes. In this paper, we further develop and extend this setting by an imposed ordering on the macro inference rules. That ordering leads to a "strict setting". It makes implementing and using variants of well-known or new schemes within this framework a very easy task even for a non-expert user. Also, this setting enables easy comparison of different combination/augmentation schemes and combination of their ideas.

**Keywords** : theorem proving, decision procedures, combining decision procedures, augmenting decision procedures

# Strict General Setting for Building Decision Procedures into Theorem Provers

Predrag Janičić[1] and Alan Bundy*[2]

[1] e-mail: janicic@matf.bg.ac.yu
Faculty of Mathematics, University of Belgrade
Studentski trg 16, 11 000 Belgrade, Yugoslavia
[2] e-mail: A.Bundy@ed.ac.uk
Division of Informatics, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, Scotland

**Abstract.** The efficient and flexible incorporating of decision procedures into theorem provers is very important for their successful use. There are several approaches for combining and augmenting of decision procedures; some of them support handling uninterpreted functions, congruence closure, lemma invoking etc. In this paper we present a variant of one general setting for building decision procedures into theorem provers (GS framework [18]). That setting is based on macro inference rules motivated by techniques used in different approaches. The general setting enables a simple describing of different combination/augmentation schemes. In this paper, we further develop and extend this setting by an imposed ordering on the macro inference rules. That ordering leads to a "strict setting". It makes implementing and using variants of well-known or new schemes within this framework a very easy task even for a non-expert user. Also, this setting enables easy comparison of different combination/augmentation schemes and combination of their ideas.

## 1 Introduction

The role of decision procedures is very important in theorem proving. Even an inefficient decision procedure can be more efficient than other techniques (e.g. induction). However, in most applications only a small number of conjectures fall within the scope of one decision procedure. For instance, the formula $\forall x \forall y\ x \leq y \wedge y \leq x + car(cons(0, x)) \wedge p(h(x) - h(y)) \rightarrow p(0)$ is not a Presburger arithmetic formula. However, if there are available decision procedures for the theory of lists with *car*, *cdr* and *cons* and for equality with uninterpreted function and predicate symbols, it is possible to combine them with a decision procedure for Presburger arithmetic and to prove the above conjecture. This example illustrates situations when several decision procedures have to *cooperate* and to be *combined* into one decision procedure. As another example, consider the formula $\forall l \forall \alpha \forall k\ (l \leq minl(\alpha) \wedge 0 < k \rightarrow l < maxl(\alpha) + k)$

* Supported in part by EPSRC grant GR/M45030.

which is not a Presburger arithmetic formula and cannot be proved by a decision procedure for that theory. However, if that decision procedure is aided by the lemma $\forall \xi\ (minl(\xi) \leq maxl(\xi))$, the above conjecture can be proved. This example illustrates how the realm of a decision procedure can be extended by the use of available lemmas. In situations like this one, a decision procedure has to communicate with other components of a prover, such as a lemma invoking mechanism, a rewriting mechanism, simplification techniques etc. Then we say we deal with *augmenting* a decision procedure (or we say we deal with *integrating* or *incorporating* a decision procedure into a theorem prover).

There are several influential approaches used in handling the problem of efficiently combining and augmenting decision procedures. The research concerning these issues has gone mostly along different two lines: one concerning combining decision procedures and one concerning augmenting decision procedures (within heuristic theorem provers). Combination schemes typically rely on some local, specific data structures and are focused on combining decision procedures [28, 32, 14, 4, 6, 31, 5], but some of them also use additional techniques (such as the use of additional rules and lemmas etc). Augmentation schemes use different functionalities of heuristic theorem provers such as rewriting techniques, lemma invoking mechanisms, a variety of simplifications etc [7, 20, 21, 1, 19, 2]. Combination schemes are typically aimed at decidable combinations of decidable theories, while augmentation schemes are primarily intended for use in undecidable extensions of decidable theories. The research on combination of decision procedures is rich in theoretical and practical results, while the research on the augmentation of decision procedures is less well developed theoretically.

Nelson/Oppen's scheme for combination of theories [27] is used in several systems including the Stanford Pascal Verifier, [24], ESC [15] and EVES. [13]. In this approach, decision procedures for disjoint theories are combined by abstracting terms which fall outside a certain theory and by propagating deduced equalities from one theory to another. Shostak's scheme for combination of theories [32] is used in several other systems including EHDM, [16], PVS, [29], STeP [25, 6] and SVC. [4]. In this approach, *solvers* for specific theories (for instance, solvers for equational real arithmetic, theory of lists etc) are tightly combined by an efficient underlying congruence closure algorithm for ground equalities.

One of the most influential methods for augmentation of decision procedures is a procedure for linear arithmetic integrated within Boyer and Moore's NQTHM [7]. In this approach, a decision procedure for Presburger arithmetic (based on Hodes' algorithm [17]) is combined with a heuristic *augmentation* which provides information about alien functions (i.e. provides lemmas) and with a technique which simplifies one literal in the clause by using the remaining literal as the context. Probably because of its informal and complicated description there are very few reimplementations of Boyer/Moore's procedure, but nevertheless, it influenced a number of other approaches and techniques (including work on augmenting an arithmetic decision procedure into a rewrite based prover TECTON [20], contextual rewriting [35], constraint contextual rewriting [1] and the EPM scheme [19]).

The general setting for building decision procedures into theorem provers (GS) [18] is aimed at capturing the key ideas shared by the above mentioned systems. It is built from a set of macro inference rules and it is flexible and general enough to cover a number of different combination/augmentation schemes. In this paper, we go one step further and extend the GS framework by an imposed ordering on the macro inference rules. While in the GS framework one can use the macro inference rules as building blocks in an (almost) arbitrary way, within the strict general setting (SGS) it is not the case. With the imposed strict ordering on the rules, it is still possible to mimic a number of combination/augmentation schemes and, on the other hand, this ordering makes different schemes very easy to implement and to use. In addition, this strict setting enables easy comparison of different combination/augmentation schemes and combination of their ideas.

*Overview of the paper* In §2 we give some basic background and notation information. In §3 we give a brief account of the GS framework. In §4 we discuss the imposed ordering on the macro inference rules and in the §5 we discuss how combination/augmentation schemes can be implemented within the SGS framework. In §6 we report on a prototype implementation of the SGS framework and on some results obtained by different schemes implemented within the framework. In §7 we discuss the related work. In §8 we discuss future work and in §9 we draw some conclusions.

## 2 Background and Notation

In this paper we deal with quantifier-free first-order theories with equality and we use many-sorted logic. We use the standard notions of formula, theory, decidable theory, extension of a theory, combination of theories, valid, satisfiable and inconsistent formula, the standard notions of term rewriting etc. For simplicity, in the rest of the paper, instead of the sorted version of equality we will write just $=$ when sorts are evident from a context or are not important.

In the rest of the paper, special attention will be given to Presburger arithmetic (according to its significance in verification problems). Presburger arithmetic is (roughly speaking) a first-order theory built up from the constant 0, variables, binary $+$, unary $s$ and relations $<$, $>$, $=$, $\leq$, $\geq$. We denote Presburger arithmetic over natural numbers by PNA, over integers by PIA and over rationals by PRA. The whole of arithmetic is undecidable, but the theories PNA, PIA, PRA are each decidable [30, 22].

For a given ordering $\prec$ on terms, a term $t$ is a *maximal* in a formula $f$ if for any other term $t'$ in formula $f$ it does not hold that $t \prec t'$. A term $t$ is called an *alien term* w.r.t. $\mathcal{T}_1, \ldots, \mathcal{T}_j$ if it is not a term of these theories. We introduce similar notions for literals.

# 3 General Setting for Building Decision Procedures into Theorem Provers

In this section we give a brief overview of the general setting for building-in decision procedures into theorem provers — the GS framework. A full and detailed account of the GS framework is given in [18].

## 3.1 Macro Inference Rules

There are several key steps in different systems for combining and augmenting decision procedures including abstraction, entailment and congruence closure. We describe them in the form of macro inference rules.

We use proof by refutation, so if $F$ is a quantifier-free formula to be proved, we need to show that $\neg F$ is unsatisfiable in $\mathcal{T}$, i.e., we need to show that $\mathcal{T} \cup \neg F$ is inconsistent. More precisely, in order to prove a quantifier-free formula $F$ in a theory $\mathcal{T}$, i.e. to show $\mathcal{T} \vdash \forall^* F$, we need to show that $\mathcal{T}, \exists^*(\neg F) \vdash \bot$ (where $\forall^*$ and $\exists^*$ denote universal and existential closure). For now on, we will assume that the theory $\mathcal{T}$ is fixed and by "$\neg F$ is unsatisfiable" we will mean "$\neg F$ is unsatisfiable in $\mathcal{T}$", etc.

We denote macro inference rules by $f_1 \Rightarrow_X f_2$, where $X$ is the name of the specific rule. If a rule $X$ is parametrised by parameters $P$ then we denote it by $X(P)$. The soundness of each macro inference rule has to be ensured; i.e., for all rules $f_1 \Rightarrow_X f_2$, it has to be ensured that if $f_2$ is unsatisfiable, then $f_1$ is unsatisfiable, too. If the inference rule $X$ is (un)satisfiability preserving, we denote it by $\Leftrightarrow_X$. We denote by $\Rightarrow^*$ the transitive closure of $\Rightarrow$ and by $\Leftrightarrow^*$ the transitive closure of $\Leftrightarrow$. If $\neg F \Rightarrow^* \bot$ then $\neg F$ is unsatisfiable, so the original formula $F$ is valid (i.e. $\mathcal{T} \vdash F$). If $\neg F \Leftrightarrow^* \top$ then there is an implicitly constructed counterexample for $F$, so it is invalid (i.e. $\mathcal{T} \nvdash F$; in a heuristic theorem prover, negative results can also be important and useful, for example, in controlling generalisation, and other non-satisfiability preserving heuristics). If it holds that $\neg F \Rightarrow^* \top$ (and does not hold that $\neg F \Leftrightarrow^* \top$), then it is *unknown* whether $F$ is valid or invalid. If $f \Leftrightarrow^* f'$ and $f' \in \{\top, \bot\}$ then we say that a corresponding proof branch is closed and $f'$ is returned as the result of the inference process. Additionally, after each macro inference step is applied, if $f \Leftrightarrow^* f'$, a formula $f'$ can be passed to any other component of a prover and some other technique (e.g. induction) can be tried.

We assume that there is available an ordering on involved function and predicate symbols that induces a reduction ordering $\prec$ on terms.

*Disjunctive normal form* If $f$ is the negation of a formula $F$ to be proved, we can transform it into disjunctive normal form and then try to refute each of its disjuncts. We denote this transformation in the following way: $f \Rightarrow_{dnf} (f_1 \vee f_2 \vee \ldots \vee f_n)$. Since transformation into disjunctive normal form is (un)satisfiability preserving, we also denote it by $\Leftrightarrow_{dnf}$.

*Proving disjuncts (case split)* If a formula $f$ is of the form $(f_1 \vee f_2 \vee \ldots \vee f_n)$ (where each $f_i$ is a conjunction of literals), in order to refute $f$, we have to refute each $f_i$. Hence,

$$f \Leftrightarrow_{split} \bot \text{ if } f_i \Rightarrow^* \bot \text{ for all } i,\ 1 \leq i \leq n\ .$$

Additionally,

$$f \Leftrightarrow_{split} \top \text{ if } f_i \Leftrightarrow^* \top \text{ for some } i,\ 1 \leq i \leq n\ .$$

We will further deal mostly with conjunctions of literals and we won't distinguish between a conjunction $l_1 \wedge l_2 \wedge \ldots \wedge l_n$ and a multiset $\{l_1, l_2, \ldots, l_n\}$.

*Simplification* Simplification is based on the following simple rules: $\{\ \} \longrightarrow \top$, $f \cup \{l\} \cup \{l\} \longrightarrow f \cup \{l\}$, $f \cup \{l\} \cup \{\neg l\} \longrightarrow \bot$, $f \cup \{\bot\} \longrightarrow \bot$, $f \cup \{\top\} \longrightarrow f$, $f \cup \{\neg(a = a)\} \longrightarrow \bot$, $f \cup \{a = a\} \longrightarrow f$, $f \cup \{x = y\} \longrightarrow f$, where $x$ is a variable that does not occur in $y$ and $f$, or $y$ is a variable that does not occur in $x$ and $f$. The inference rule which exhaustively applies the above rules we denote by $\Leftrightarrow_{simpl}$. In addition, there are simplifications specific to each theory. For instance, a literal $x \leq (y + z) - (y - x)$ can be simplified to $0 \leq z$ by a simplifier for PRA. Also, a simplifier for a theory $\mathcal{T}_i$ should detect valid and invalid ground $\mathcal{T}_i$ literals and rewrite them to $\top$ and $\bot$, respectively. Simplification for a specific theory $\mathcal{T}_i$ is performed only on $\mathcal{T}_i$ literals. Dealing with a combination of theories $\mathcal{T}_i$ $(i = 1, 2, \ldots, k)$, we iteratively use simplifications for theories $\mathcal{T}_i$ $(i = 1, 2, \ldots, k)$ and we denote this simplification (the extended form of $\Leftrightarrow_{simpl}$) by $\Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$.

*Abstraction* Depending on the dominating functional and relational symbols, we can conditionally consider that an atomic formula belongs to a theory $\mathcal{T}_i$ and abstract its alien terms (by new, *abstraction variables*) in order to obtain an atomic formula of a theory $\mathcal{T}_i$. We can perform abstraction for several theories $\mathcal{T}_1$, ..., $\mathcal{T}_k$ in one step (in such a way that each abstracted literal belongs to one of $\mathcal{T}_1$, ..., $\mathcal{T}_k$). We can replace all original literals by their abstracted versions (which is not (un)satisfiability preserving): $f \Rightarrow_{abs(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)} f'$. We can also add the set $C$ of equations defining newly introduced variables and we denote that by $f \Leftrightarrow_{abs^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)} f' \cup C$. Abstraction can be propagated if subterms of abstracted terms still have alien terms with respect to the theories $\mathcal{T}_1$, $\mathcal{T}_2$,...,$\mathcal{T}_k$. This form of abstraction always adds equations $C$ defining newly introduced variables and we denote it by $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$.

*Replacement* If a current formula $f$ is a conjunction of literals and if there is a literal $x = t$ where $x$ is a variable and $t$ is a term which does not include occurrences of $x$, then we can replace all occurrences of $x$ in all formulae by $t$ and delete the literal $x = t$. We can perform this operation for all variables $x$ fulfilling the given condition one after another (for simplicity, we don't discuss the order of variables and its effect) obtaining a formula $f'$ and we denote this inference by: $f \Leftrightarrow_{repl} f'$. Another version of the replacement is restricted only to replacement of variables by variables (i.e., in the case where there are equalities

of the form $x = y$); such replacement eliminates one variable and does not introduce new alien terms. We denote it by $\Leftrightarrow_{repl^=}$.

*Unsatisfiability* Let us suppose that there is available a procedure which can detect if a conjunction of some literals $\{l_{i_1}, l_{i_2}, \ldots, l_{i_k}\}$ $(i_k \leq n)$ from a formula $f$ is unsatisfiable in a theory $\mathcal{T}_i$, where $f$ is a conjunction of literals $l_1, l_2, \ldots, l_n$. If the conjunction $\{l_{i_1}, l_{i_2}, \ldots, l_{i_k}\}$ $(i_k \leq n)$ is unsatisfiable, then $f$ is unsatisfiable, too. We denote this inference in the following way: $f \Leftrightarrow_{unsat(\mathcal{T}_i)} \bot$.

*Entailment* Let $f$ be a disjoint union of two sets of literals: $A$ and $B$. Let us suppose that literals $B = \{l_{i_1}, l_{i_2}, \ldots, l_{i_k}\}$ entail (in a theory $\mathcal{T}_i$) some conjunction or disjunction of literals $C$ (which does not occur in $A \cup B$). We assume that soundness is guaranteed by the specification (which is denoted by $P$ for $\mathcal{T}_i$) of this entailment. If we keep the literals from $B$, we denote it by $A \cup B \Leftrightarrow_{entail^+(P,\mathcal{T}_i)} A \cup B \cup C$. If we don't keep the literals from $B$, we denote it by $A \cup B \Rightarrow_{entail(P,\mathcal{T}_i)} A \cup C$, while in some cases it also holds that $A \cup B \Leftrightarrow_{entail(P,\mathcal{T}_i)} A \cup C$. Examples of entailment are:

- Nelson/Oppen style entailment [27, 34]: $f \Leftrightarrow_{entail^+(NO,\mathcal{T}_i)} f \cup \{\bigvee_{i=1}^{k} x_i = y_i\}$ This sort of entailment (used in Nelson/Oppen's procedure [27]) yields an implicit equation ($k = 1$), or, in case of *non-convex*[1] theory, a disjunction of equalities.
- Shostak style entailment: if $\mathcal{T}$ is algebraically solvable $\sigma$–theory and if $solve(e) \equiv E$ then we denote this kind of entailment (used in Shostak's procedure [32, 14]) by $A \cup \{e\} \Leftrightarrow_{entail(solve,\mathcal{T})} A \cup \{E\}$.
- Fourier/Motzkin's entailment: in the TECTON system [20] there is used a procedure (inspired by Fourier/Motzkin's method [23]) for entailing implicit equalities $E$ from Presburger rational arithmetic (PRA) inequalities $I$. We denote it by $A \cup I \Leftrightarrow_{entail^+(impl-eqs,pra)} A \cup I \cup E$.
- Variable elimination: let us assume that there is available a quantifier elimination procedure $P$ for a theory $\mathcal{T}$ (which might serve as the basis for a decision procedure for $\mathcal{T}$), i.e., a procedure which transforms a formula $f$ (of a theory $\mathcal{T}$) into an equivalent formula $f'$ with fewer variables. Let us suppose that there is a subset $B$ of given literals $F$ ($F = A \cup B$) such that: $(i)$ all literals from $B$ belong to $\mathcal{T}$; $(ii)$ there is a variable $x$ such that it does not occur in $A$. Then we can perform a variable elimination procedure $P$ on the literals $B$ and on the variable $x$, yielding a formula $C$ (note that $C$ may contain disjunctions). Note that, in our setting, all variables are (implicitly) existentially quantified (recall that we deal only with quantifier-free formulae) and, thus, it is sufficient to have available only elimination of existentially quantified variables. We denote elimination of all variables fulfilling the given conditions by: $A \cup B \Rightarrow_{entail(P,\mathcal{T})} A \cup C$, or, for some procedures and theories, by $A \cup B \Leftrightarrow_{entail(P,\mathcal{T})} A \cup C$. Some instances

[1] A formula $F$ is *non-convex* if $F$ entails $x_1 = y_1 \vee x_2 = y_2 \vee \ldots x_n = y_n$, but for no $i$ between 1 and $n$ does $F$ entail $x_i = y_i$. Otherwise, $F$ is *convex*. A theory is *convex* if every conjunction of its literals is convex. Otherwise, it is *non-convex*.

of variable elimination entailment are: $\Leftrightarrow_{entail(Cooper,pia)}$, $\Leftrightarrow_{entail(Hodes,pra)}$, $\Leftrightarrow_{entail(Fourier,pra)}$, $\Rightarrow_{entail(Hodes,pia)}$. Fourier's method for variable elimination is used in TECTON [20] and is essentially the same as Hodes' procedure used in NQTHM. Note that Hodes' [17] procedure is an (un)satisfiability preserving procedure for PRA, but not for PIA. Cooper's procedure [12] is (un)satisfiability preserving for PIA and PNA.

The entailment rule, typically, has an essential role in combination/augmentation schemes.

*Constant Congruence Closure/Ground Completion* Constant congruence closure and ground completion are substantially the same thing [3]. In our general setting, we perform a constant congruence closure on a set of all equalities in a formula being proved. This returns a set of equivalence classes; from each class we choose a minimal element (according to an ordering $\prec$) and introduce equalities for all remaining terms in that class. The equalities are then oriented as rewrite rules (according to the ordering $\prec$) and inter-reduced (while trivial equalities are eliminated). These oriented equalities make a ground canonical system (note that we treat variables as constants). They replace the initial set of equalities and are further used to normalise other literals in a formula being proved (literals other than equalities). In addition, if in an obtained formula there is detected a contradiction in the pure theory of equality (i.e. if there is a literal $\neg(t = t)$ or literals $l$ and $\neg l$) this rule returns $\bot$. This transformation we call simply *constant congruence closure* and we denote it by $\Leftrightarrow_{ccc}$.

*Lemma Invoking* We assume that there may be available some additional rewrite rules, definitional equalities, additional theorems and lemmas all of which we treat in the same way.[2] We consider only conditional rewrite rules (i.e., rules of the form $l \rightarrow r$ **if** $p_1, p_2, \ldots, p_k$ and $\rho \rightarrow \top$ **if** $p_1, p_2, \ldots, p_k$). We assume (in order to ensure termination) that there is available a reduction ordering $\prec$ such that for each rule $l \rightarrow r$ **if** $p_1, p_2, \ldots, p_k$ it holds that $r \prec l$ and $p_i \prec l$ (for $i = 1, 2, \ldots, k$) and for each rule $\rho \rightarrow \top$ **if** $p_1, p_2, \ldots, p_k$ it holds that $p_i \prec \rho$ (for $i = 1, 2, \ldots, k$). If our proving strategy is based on a decision procedures for some theories $\mathcal{T}_1, \ldots, \mathcal{T}_j$ we assume that the available additional rules make a consistent conservative extension of each of them. Then it is only sensible to search for lemmas which are not formulae of these theories and we formulate the lemma invoking inference with respect to some theories $\mathcal{T}_1, \ldots, \mathcal{T}_j$.

[2] Note that these additional rewrite rules does not need necessarily to be lemmas i.e. theorems of the underlying theory; within our system we don't examine the status of these rules and therefore it would be more appropriate to consider these rules as additional hypotheses. Hence, if this macro inference rule (with the set $\mathcal{L}$ of additional rules) is used in proving a formula $F$, then we have to write $\mathcal{T}, \mathcal{L} \vdash F$ (rather than $\mathcal{T} \vdash F$). However, because additional rules are often indeed lemmas (in a strict sense) of the underlying theory and because of tradition reason, we also use the terminus "lemma invoking" for this rule and for dealing with all of the additional rules.

Let $t$ be a maximal alien term with respect to a set of theories $\mathcal{T}_1, \ldots, \mathcal{T}_j$ (by analogy we define the rule with respect to a maximal alien literal) in the formula $f$ being refuted (and there is no alien literal $l$ such that $t \prec l$). Let $c$ be a new abstraction variable for the term $t$ (if there is no equality $t = c$ in the formula already). Let us assume that there is a rule $\rho \rightarrow \top$ **if** $p_1, p_2, \ldots, p_k$ available in the set $\mathcal{L}$ such that: a maximal alien term in $\rho$ (w.r.t. $\mathcal{T}_1, \ldots, \mathcal{T}_j$) is a term $t'$ and there is a most general substitution $\phi$ such that $t'\phi$ is equal to $t$. Also, let us assume that all variables occurring in the lemma are instantiated by $\phi$. The formula $f$ is transformed into $f \wedge (p_1 \wedge p_2 \wedge \ldots \wedge p_k \Rightarrow \rho)\phi$ and in this formula all occurrences of $t$ are replaced by $c$, yielding a resulting formula $f'$ (and leading to a case split if $k > 0$). If there is no such lemma then all occurrences of $t$ are replaced by $c$ yielding the resulting formula $f'$. This inference we denote by: $\Rightarrow_{lemma^+(\mathcal{L},\mathcal{T}_1,\ldots,\mathcal{T}_j)}$.

If there is more than one lemma fulfilling the required conditions, then in the case of failing to prove the unsatisfiability, one can backtrack and try with another lemma. Even if all lemmas are unsuccessfully tried, one can try to proceed with the refuting process with no help from any lemma (for some conjectures with alien terms, no lemmas are required — for instance, $\forall\alpha\ \forall k\ \ (max(\alpha) \leq k \vee max(\alpha) > k)$ can be proved without any lemma). In [20, 21, 1], when using conditional rules, for each condition there is one subproof and this leads to tree-structured proofs. It is analogous to the lemma invoking mechanism proposed here, since for the conclusion and for each condition from the used conditional rule, there is one disjunction (after $\Leftrightarrow_{dnf}$ and $\Leftrightarrow_{split}$ rules applied) that has to be refuted. Note that that we usually leave the realm of completeness when we start to rely on lemmas [35, 20, 21, 19].

### 3.2 Implementing combination/augmentation schemes within GS framework

A number of combination/augmentation schemes can be represented/implemented within the GS framework built from the above macro inference rules, including Nelson/Oppen's, Shostak's and Kapur/Nie's schemes. These representations do not always fully represent the original methods[3] and their efficiency, but give their key ideas. As an illustration, we give a description of a Nelson/Oppen style procedure. Let quantifier-free theory $\mathcal{T}$ be a combination of theories $\mathcal{T}_i$ $(1 \leq i \leq k)$ which do not share non-logical symbols other than equality. Let $F$ be a formula to be proved. The procedure is as follows ($\neg F$ is its input):

1. Apply $\Leftrightarrow_{dnf}$
2. Apply $\Leftrightarrow_{split}$
3. Apply $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$
4. Apply $\Leftrightarrow_{unsat(\mathcal{T}_i)}$, for some $i$, $1 \leq i \leq k$
5. Apply $\Leftrightarrow_{entail^+(NO,\mathcal{T}_i)}$, for some $1 \leq i \leq k$; if not successful, return $\top$.

[3] This is especially the case with the implementation of Shostak's scheme, whose original version is very compact and optimised.

6. Apply $\Leftrightarrow_{repl=}$
7. Go to step 1

In the given description we use the terminus "apply [an inference rule]", while it would be more precise to say "try to apply [an inference rule]". Note that if the theories $\mathcal{T}_i$ are convex, then in step 7, we can jump to step 4, instead of to step 1 [27, 34]. In addition, since no rule introduces new alien terms, it is sufficient to apply the rule $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,...,\mathcal{T}_k)}$ only once, but, for simplicity, we won't try to optimise its use. The replacement carried by the rule $\Leftrightarrow_{repl=}$ (not used in the original procedure) is useful since it reduces the number of considered variables in the search for implicit equalities. Note that returning $\top$ in step 5 is not covered by the introduced macro inference rule — it is a kind of meta level inference. Under certain conditions, it can be proved that if the above procedure returns $\top$, a given conjecture is invalid: moreover, if $\mathcal{T}_i$ $(1 \leq i \leq k)$ are stably-infinite, signature disjoint, quantifier-free theories, then the above, Nelson/Oppen style procedure terminates and is complete and sound [18].

## 4 Ordering on macro inference rules and SGS framework

Merged with each other, a number of (variants of) different known schemes implemented within the proposed general setting have similar structures. Moreover, some slight changes can be made in order to have these structures clearly comparable in the sense that similar rules are in the same or similar positions. By this motivation, we impose a fixed ordering on the macro inference rules on the basis of a number of known schemes and schemes implemented within the GS framework. This ordering makes a basis for a strict general setting (SGS) for building decision procedures into theorem provers. A combination/augmentation scheme is, within the SGS framework, simply created by choosing and (de)activating certain rules at their fixed positions. The imposed ordering on the rules is as follows:

1. *dnf* rule $\Leftrightarrow_{dnf}$
2. *case split* rule $\Leftrightarrow_{split}$
3. simplification:
   - $\Leftrightarrow_{simpl}$
   - $\Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,...,\mathcal{T}_k)}$
4. constant congruence closure $\Leftrightarrow_{ccc}$
5. abstraction:
   - $\Rightarrow_{abs(\mathcal{T}_1,\mathcal{T}_2,...,\mathcal{T}_k)}$
   - $\Leftrightarrow_{abs^+(\mathcal{T}_1,\mathcal{T}_2,...,\mathcal{T}_k)}$
   - $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,...,\mathcal{T}_k)}$
   - $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,...,\mathcal{T}_k,\mathcal{E})}$
6. unsatisfiability $\Leftrightarrow_{unsat(\mathcal{T}_i)}$
7. entailment:
   - $\Rightarrow_{entail(A,\mathcal{T}_i)}$
   - $\Leftrightarrow_{entail(A,\mathcal{T}_i)}$

   - $\Leftrightarrow_{entail^+(A,\mathcal{T}_i)}$
8. replacement:
   - $\Leftrightarrow_{repl^=}$
   - $\Leftrightarrow_{repl}$
9. lemma invoking $\Rightarrow_{lemma^+(\mathcal{L},\mathcal{T}_1,\mathcal{T}_2,...,\mathcal{T}_j)}$
10. jump — Go to start

Some rules are parametrised by a set of theories which determines an instance of a combination/augmentation scheme. In some schemes, it is convenient to give the pure theory of equality a special treatment and so in the above list we give the following special case of abstraction rule: $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,...,\mathcal{T}_k,\mathcal{E})}$. In principle, we should also list versions of this kind for $\Rightarrow_{abs(\mathcal{T}_1,\mathcal{T}_2,...,\mathcal{T}_k)}$ and $\Leftrightarrow_{abs^+(\mathcal{T}_1,\mathcal{T}_2,...,\mathcal{T}_k)}$, but we haven't found them very useful. We also include a rule "Go to start" into the set of the rules.

## 5 Implementing combination/augmentation schemes within SGS framework

A scheme within the SGS framework is represented by a sequence of the macro inference rules used (i.e. by a corresponding sequence of 0s and 1s). In addition, some rules (entailment rules) have to be instantiated by specific underlying algorithms; for some schemes and some rules, if the rule fails, the scheme has to return $\top$; for some schemes and some rules, if the rule is successful, a control has to go to the start of the scheme. For instance, if in an entailment rule we use Nelson/Oppen style entailment ($\Leftrightarrow_{entail^+(NO,\mathcal{T}_i)}$), if the rule has to return $\top$ when unsuccessful and if the rule does not lead to a start of a scheme when successful, we denote these additional parameters in the following way: $(NO; 1; 0)$. We omit these additional parameters if they have default values (the default values are $(/; 0; 0)$). For instance, Nelson/Oppen style scheme described in 3.2 is represented in the following way: $1100000101001(NO; 1; 0)1001$. Table 1 gives representations of several combination/augmentation schemes: schemes made in the style of Nelson/Oppen's, Shostak's, TECTON (we discuss only one procedure from [20] — one concerning PRA) and EPM procedures. Note that it is sensible to use only one variant of the abstraction rule, only one variant of the simplification and only one variant of the replacement in one scheme. However, it may be sensible to use more variants of the entailment rule (as in TECTON).

A combination/augmentation scheme is additionally specified by a set $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k\}$ of the underlying theories involved. In the Nelson/Oppen scheme, an input formula belongs to the union of the underlying theories. In the Shostak style schemes, an input formula belongs to the union of the underlying theories and the pure theory of equality ($\mathcal{E}$). In TECTON and the EPM style scheme $k = 1$ and an input formula belongs to an (conservative) extension of the theory $\mathcal{T}_1$. In the TECTON style scheme $\mathcal{T}_1$ is PRA. In the EPM style scheme $\mathcal{T}_1$ is a theory which admits quantifier elimination (e.g. PRA, PIA) and $A$ is a quantifier elimination procedure for $\mathcal{T}_1$ (e.g. Hodes' procedure for PRA). If in the EPM

| # | Rule | N/O style | Shostak style | TECTON style | EPM style |
|---|---|---|---|---|---|
| 1. | $\Leftrightarrow_{dnf}$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ |
| 2. | $\Leftrightarrow_{split}$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ |
| 3a. | $\Leftrightarrow_{simpl}$ | | $\surd$ | | |
| 3b. | $\Leftrightarrow_{simpl(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$ | | | $\surd$ | $\surd$ |
| 4. | $\Leftrightarrow_{ccc}$ | | $\surd$ | $\surd$ | |
| 5a. | $\Rightarrow_{abs(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$ | | | | |
| 5b. | $\Leftrightarrow_{abs^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$ | | | | |
| 5c. | $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$ | $\surd$ | | | $\surd$ |
| 5d. | $\Leftrightarrow_{absp^+(\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k,\mathcal{E})}$ | | $\surd$ | $\surd$ | |
| 6. | $\Leftrightarrow_{unsat(\mathcal{T}_i)}\ (i=1,\ldots,k)$ | $\surd$ | | $\surd$ | $\surd$ |
| 7a. | $\Rightarrow_{entail(A,\mathcal{T}_i)}\ (i=1,\ldots,k)$ | | | | |
| 7b. | $\Leftrightarrow_{entail(A,\mathcal{T}_i)}\ (i=1,\ldots,k)$ | | $\surd\ (solve;1;0)$ | $\surd\ (Fourier;0;1)$ | $\surd\ (A;0;0)$ |
| 7c. | $\Leftrightarrow_{entail^+(A,\mathcal{T}_i)}\ (i=1,\ldots,k)$ | $\surd\ (NO;1;0)$ | | $\surd\ (impl\text{–}eqs;0;1)$ | |
| 8a. | $\Leftrightarrow_{repl=}$ | $\surd$ | | | |
| 8b. | $\Leftrightarrow_{repl}$ | | $\surd$ | | $\surd$ |
| 9. | $\Leftrightarrow_{lemma^+(\mathcal{L},\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_k)}$ | | | $\surd$ | $\surd$ |
| 10. | Go to start | $\surd$ | $\surd$ | $\surd$ | $\surd$ |

**Table 1.** Descriptions of some combination/augmentation schemes within the SGS framework (underlying theories are $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$).

scheme a procedure which does not preserve unsatisfiability is used (e.g. Hodes' procedure for PIA) we choose the rule $\Rightarrow_{entail(A,\mathcal{T}_i)}$ (7a) instead of $\Leftrightarrow_{entail(A,\mathcal{T}_i)}$ (7b).

All of the represented schemes are sound (because all the macro inference rules are sound) and terminating. Only the Nelson/Oppen style scheme is complete in its scope. It was recently shown by Rueß and Shankar that the original Shostak's algorithm is not complete [31]; the same holds for the Shostak style scheme represented above. TECTON and the EPM scheme rely on lemmas and, as typical for augmentation schemes, they are not complete.

Instead of a number of implemented combination/augmentation schemes, within the SGS framework there is only one meta-scheme. It is parametrised by a specification for a scheme and by a set of underlying theories. There are no scheme-specific parts built-in or invoked from the meta-scheme. The set of theories can be determined for each conjecture by a grammars library mechanism. This mechanism, for instance, tries to find a (small) set of available theories such that a conjecture belongs to their union; if for each of them there is available NO entailment, then the Nelson/Oppen style scheme can be applied. Similarly, this mechanism can try to find if a conjecture belongs to an extension of some theory which admits quantifier elimination, while there is available a corresponding quantifier elimination procedure and the extension is given by some additional rules and lemmas; if so, EPM style procedure can be applied. The ordering of schemes can be adjusted according to different criteria (for certain theories, the

Nelson/Oppen style scheme is complete, but, for instance, the TECTON style scheme is often more efficient).

## 6 Prototype implementation and results

We have implemented the SGS framework in SWI Prolog, within the *Clam* system [8] and on top of the implementation of the GS framework described in [18]. That implementation is not the most efficient possible (e.g. the rule $\Leftrightarrow_{ccc}$ is implemented on the basis of the simpler Nelson/Oppen's algorithm and not on the basis of the more efficient, Shostak's algorithm; Nelson/Oppen's entailment is implemented on the basis of the connection: $f \Leftrightarrow_{entail^+(NO,\mathcal{T}_i)} f \cup \{x = y\}$ if and only if $f \cup \{\neg(x = y)\} \Leftrightarrow_{unsat(\mathcal{T}_i)} \bot$ and not on the basis of some specific properties of the theories involved; the rule $\Rightarrow_{entail^+(NO,pra)}$ is used instead of $\Rightarrow_{entail^+(impl-eqs,pra)}$ etc). However, even in this implementation, based on only few underlying procedures, the framework gives satisfactory results (in terms of CPU time).

| # | Conjecture | N/O | Shostak | TECTON | EPM |
|---|---|---|---|---|---|
| 1. | $x \leq y, y \leq x + car(cons(0,x)), p(h(x) - h(y)), \neg p(0)$ [27] | 0.70 | n/i | ? | ? |
| 2. | $z = f(x - y), x = y + z, \neg(y + f(f(z)) = x)$ [32] | 0.19 | 0.14 | 0.28 | ? |
| 3. | $x = y, f(f(f(x))) = f(x), \neg(f(f(f(f(f(y))))) = f(x))$ [14] | 0.01 | 0.01 | 0.01 | ? |
| 4. | $y = f(z), x - 2 \cdot y = 0, \neg(f(x - y) = f(f(z)))$ [14] | 0.14 | 0.13 | 0.09 | ? |
| 5. | $f(x) = 4, f(2 \cdot y - x) = 3, x = f(2 \cdot x - y)$, $4 \cdot x = 2 \cdot x + 2 \cdot y$ [14] | 0.65 | 0.51 | 0.28 | ? |
| 6. | $f(a - 1) - 1 = a + 1, f(b) + 1 = b - 1, b + 1 = a$ [31] | 0.84 | ? | 0.51 | ? |
| 7. | $f(a) = a, f(b) = b - 1, a = b$ [31] | 0.03 | 0.03 | 0.04 | 0.03 |
| 8. | $p(a), l \leq f(max(a,b)), 0 < min(a,b), a \leq max(a,b)$ $max(a,b) \leq a, \neg(l < g(a) + b)$ [20] lemma: $p(x) \Rightarrow f(x) \leq g(x)$ lemma: $max(x,y) = x \Rightarrow min(x,y) = y$ | ? | ? | 2.33 | ? |
| 9. | $l \leq minl(\alpha), 0 < k, maxl(\alpha) + k \leq l$ [7] lemma: $minl(\xi) \leq maxl(\xi)$ | ? | ? | 0.23 | 0.19 |
| 10. | $lp + lt \leq maxint, i \leq lt, \neg(i + \delta(pat, lp, c) \leq maxint)$ [7] lemma: $\delta(x,y,z) \leq y$ | ? | ? | 0.24 | 0.10 |

**Table 2.** Comparison between the schemes for combining/augmenting decision procedures implemented within the general setting (*n/i* means "not implemented"; *?* means that the scheme terminated on the conjecture, but failed to prove or disprove it; CPU time is given in seconds.)

The implementation of the SGS framework consists of a meta-scheme and of a control mechanism which uses representations of specific schemes and sets of theories as parameters. Some results obtained on examples from the literature and on the schemes described in §5 are given in Table 2 (tests were made on a PC 433MHZ 64MB, running under LINUX). (Note that the Shostak style procedure

fails to refute the conjunction $f(v-1)-1 = v+1, f(u)+1 = u-1, u+1 = v$ which also witnesses the incompleteness of the original Shostak's algorithm [31].) These results give just as an illustration of efficiency of the combination/augmentation schemes implemented within the SGS framework, while further tests on larger corpora are needed.

The given results show that each of the implemented scheme is most successful on some of the examples. This suggests that it is sensible to have more combination/augmentation schemes available in one theorem prover and to choose between them on the basis of some heuristic scores. These scores can be dynamically adjusted according to theorems already successfully proved (see, for instance, [26]). The SGS system with one general meta-scheme, seems a good framework for providing possibilities for using different combination/augmentation scheme in a uniform manner.

## 7 Related Work

The long line of research concerning combining and augmenting decision procedures is related to the GS framework and its extension discussed in this paper. Several systems described in the literature and used in practice were the basis for introducing the given macro inference rules [27, 32, 7, 20, 1]. In the late 90s there were introduced several new approaches aimed at giving a more general, uniform view to different combining or augmenting strategies. We briefly discuss three such approaches: constraint contextual rewriting, the EPM scheme and one flexible framework for cooperating decision procedures. Armando and Ranise's constraint contextual rewriting (CCR) [1, 2] is a formal system motivated by the ideas from Boyer and Moore's system and aimed at the flexible augmentation/integration of decision procedures into theorem provers. CCR is an extended form of contextual rewriting [35] which incorporates the functionalities provided by a decision procedure. It provides a flexible framework which can be instantiated by a specific decision procedure $X$ for some theory and it can cover approaches used by Boyer and Moore and by Kapur and Nie (or, at least, their essential parts). The soundness and termination of CCR is proved formally for the abstract scheme when certain requirements on the rewriting mechanism and the decision procedure are satisfied [2]. The EPM scheme [19] is in spirit similar to CCR — it provides a flexible framework for extending the realm of one decision procedure for a theory $\mathcal{T}$ (which admits quantifier elimination) by the use of available lemmas. The framework can be used for different theories and for different decision procedures. A new procedure can be simply "plugged in" to the system only if it provides a small set of functionalities such as checking satisfiability and elimination of quantifiers. The soundness and termination of EPM is proved for the abstract framework, given a decision procedure which provides the needed functionalities. Barrett, Dill and Stump's work [5] gives a flexible framework for cooperating decision procedures. In this framework (built on a few abstract primitive methods), Nelson/Oppen and Shostak style procedures can be implemented and their correctness (termination, soundness and complete-

ness) are formally proved. The GS framework and its extension SGS discussed in this paper are more general than the above approaches as the GS and SGS frameworks address both the problem of combining decision procedures and the problem of augmenting decision procedures. We are not aware of any previous work on superschemes that can be instantiated to different combination and augmentation schemes. The relationship between combination/augmentation schemes discussed in this paper is illustrated in Figure 1 (contextual rewriting is not a combination/augmentation scheme, but is closely related to them).

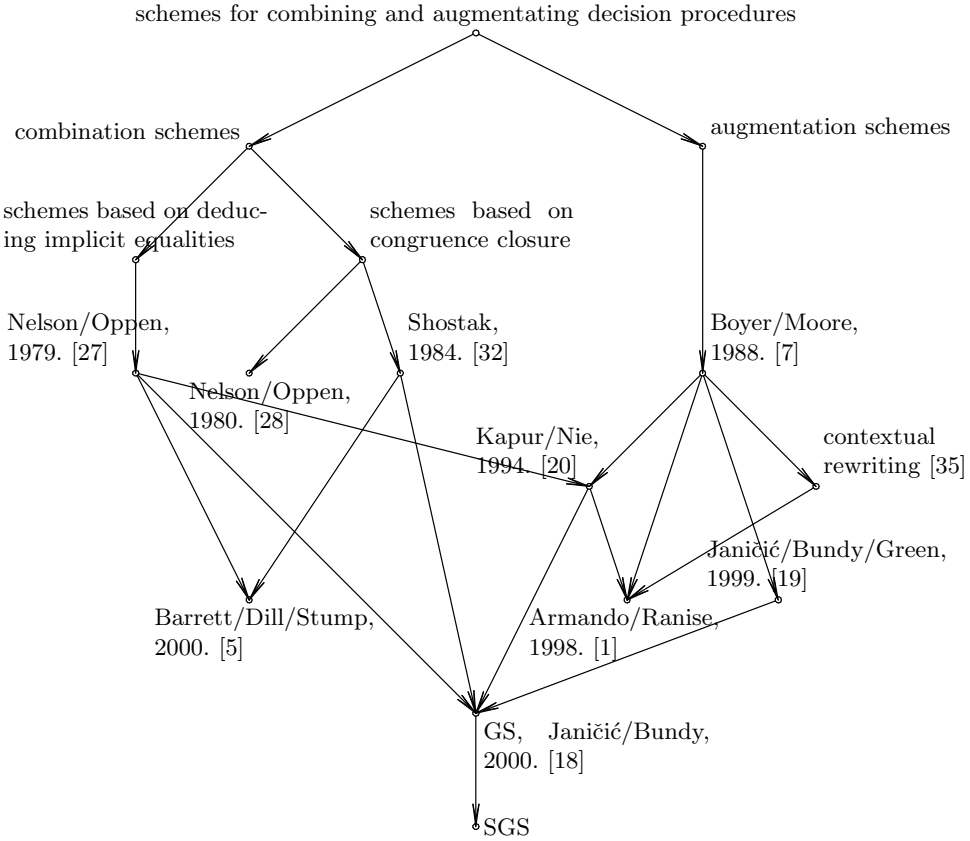


**Fig. 1.** Schemes for combining and augmenting decision procedures and their relationship

## 8 Future Work

In future work, we will investigate possibilities for proving termination of different schemes implemented within the SGS framework. Such an extension of the framework would be based on some further properties imposed on all rules; it would serve as a basis for some well-founded ordering of formulae and, hopefully, lead to a generic termination proof. Proving completeness for more schemes at the same time is an analogous but even more challenging problem.

We will investigate possibilities for making SGS schemes which combine scopes of some other SGS schemes; e.g. we will investigate possibilities for combining the scopes of the schemes described in this paper. There might be difficulties in achieving this as those schemes are defined for different classes of theories. We will also investigate possibilities for automatic generating of different instantiations of the specific macro inference rules. We will try to make a mechanism which for a new theory recognises relevant available rewrite rules and uses them in implementing specific macro inference rules for that theory — such as simplification, quantifier elimination etc. That approach follows the ideas from [11].

We will try to extend the SGS system over the quantifier-free fragment. Also, we will try to refine the lemma invoking mechanism in situations in which there are no available reduction ordering or additional rules cannot be oriented.

We are planning to investigate and use more efficient versions of specific macro inference rules. More efficient implementations of the macro inference rules would lead to much more efficient schemes. We are planning to fully integrate the SGS framework in the proof-planning system *Clam*. We believe that the proof-planning paradigm [10, 9] is a convenient environment for the SGS framework. The macro inference rules implemented as methods are used by the meta-scheme represented as a supermethod. Theories would be represented by their signatures, by their properties (e.g. convex, $\sigma$-theory, admits quantifier elimination etc) and by available primitive procedures (e.g. checking unsatisfiability, solve, variable elimination etc). Grammar library mechanisms will check the applicability of each scheme by checking the available theories and will determine the set of underlying theories (as a parameter for the scheme). If more than one scheme is applicable, they would be ordered by some heuristic scores which would be dynamically adjusted according to theorems already successfully proved.

## 9 Conclusions

In this paper we introduce a general framework for building decision procedures into theorem provers. It is an extension of the GS framework [18] which is formed from a fixed set of macro inference rules. These rules are based on the key ideas of different combination/augmentation schemes. The SGS framework provides a way of implementing and using a number of schemes in a uniform way. Schemes implemented within the SGS framework give simple and easy to read proofs. On the one hand, the SGS framework is very flexible (as all rules are modular and each rules can be independently implemented/changed/improved as long as it

meets its specification), and on the other hand it is very rigid (as all macro inference rules have their fixed positions and they can be just activated or deactivated). All this makes the SGS framework easy to implement and easy to use. On the basis of only few macro inference rules implemented, a number of combination/augmentation schemes can be obtained. It is easy to explore different variants of different schemes even for a non-expert user. We believe that the SGS framework can be used in a wide spectrum of theorem provers as a main or the only mechanism for using decision procedures. Concerning efficiency, schemes implemented within the SGS framework are typically less efficient that some tightly integrated procedures, but we believe that potential losses in efficiency will be dominated by the advantages of the SGS framework. We have implemented a prototype version of the SGS framework described and have successfully proved a number of conjectures. We are planning to use this implementation in the proof-planning system *Clam*.

## References


1. A. Armando and S. Ranise. Constraint Contextual Rewriting. In *Proceedings of the International Workshop on First order Theorem Proving (FTP'98)*, pages 65–75, Vienna, Austria, November, 23-25 1998.
2. A. Armando and S. Ranise. Termination of Constraint Contextual Rewriting. In *Proceedings of 3rd International Workshop on Frontiers of Combining Systems (FroCoS'2000)*, Lecture Notes in Artificial Intelligence 1794, pages 47–61, Nancy, France, March 2000.
3. L. Bachmair and A. Tiwari. Abstract Congruence Closure and Specializations. In David A. MacAllester, editor, *CADE–17*, number 1831 in Lecture Notes in Artificial Intelligence. Springer.
4. C. Barrett, D. Dill, and J. Levitt. Validity Checking for Combinations of Theories with Equality. In *International Conference on Formal Methods in Computer-Aided Design*, number 1166 in LNCS, pages 187–201. Springer, 1996.
5. Clark W. Barrett, David L. Dill, and Aaron Stump. A Framework for Cooperating Decision Procedures. In David A. MacAllester, editor, *CADE–17*, number 1831 in Lecture Notes in Artificial Intelligence. Springer, 2000.
6. N. S. Bjørner. *Integrating decision procedures for temporal verification*. PhD thesis, Stanford University, 1998.
7. R. S. Boyer and J S. Moore. Integrating decision procedures into heuristic theorem provers: A case study of linear arithmetic. In J. E. Hayes, J. Richards, and D. Michie, editors, *Machine Intelligence 11*, pages 83–124, 1988.
8. A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, editor, *Proceedings of the 10th Conference on Automated Deduction*, number 449 in Lecture Notes in Artificial Intelligence, pages 647–648. Springer-Verlag, 1990. Also available from Edinburgh as DAI Research Paper 507.
9. Alan Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In R. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction*, pages 111–120. Springer-Verlag, 1988. Longer version available from Edinburgh as DAI Research Paper No. 349.
10. Alan Bundy. A science of reasoning. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991. Also available from Edinburgh as DAI Research Paper 445.

11. Alan Bundy. The Use of Proof Plans for Normalization. In R. S. Boyer, editor, *Essays in Honor of Woody Bledsoe*, pages 149–166. Kluwer, 1991. Also available from Edinburgh as DAI Research Paper No. 513.
12. D. C. Cooper. Theorem proving in arithmetic without multiplication. In B. Meltzer and D. Michie, editors, *Machine Intelligence 7*, pages 91–99. Elsevier, New York, 1972.
13. D. Craigen, S. Kromodimoeljo, I. Meisels, B. Pase, and M. Saaltink. EVES: An overview. In *Proceedings of Formal Software Development Methods (VDM '91)*, number 552 in LNCS, pages 389–405. Springer, 1991.
14. D. Cyrluk, P. Lincoln, and N. Shankar. On Shostak's Decision Procedure for Combinations of Theories. In M. A. McRobbie and J. K. Slaney, editors, *Proceedings of the 13th Conference on Automated Deduction*, number 1104 in Lecture Notes in Artificial Intelligence. Springer, 1996.
15. Dave Detlefs. An overview of the extended static checking system. In *Proceedings of the First Workshop on Formal Methods in Software Practice*, pages 1–9. ACM (SIGSOFT), 1996.
16. Ehdm. User Guide for the EHDM Specification Language and Verification System, Version 6.1. Technical report, Computer Science Laboratory, SRI International, Menlo Park,CA., 1993.
17. Louis Hodes. Solving problems by formula manipulation in logic and linear inequalities. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, Imperial College, London, England, 1971.
18. Predrag Janičić and Alan Bundy. A general setting for the flexible combining and incorporating decision procedures into theorem provers. 2000. Submitted to *Journal of Automated Reasoning*.
19. Predrag Janičić, Alan Bundy, and Ian Green. A framework for the flexible integration of a class of decision procedures into theorem provers. In Harald Ganzinger, editor, *CADE–16*, number 1632 in Lecture Notes in Artificial Intelligence Series, pages 127–141. Springer, 1999.
20. Deepak Kapur and Xumin Nie. Reasoning About Numbers in Tecton. In *Proceedings of 8th International Symposium on Methodologies for Intelligent Systems, (ISMIS'94)*, pages 57–70, Charlotte, North Carolina, October 1994.
21. Deepak Kapur and M. Subramaniam. Using an Induction Prover for Verifying Arithmetic Circuits. *Software Tools for Technology Transfer*, 1999.
22. Georg Kreisel and Jean Louis Krivine. *Elements of mathematical logic: Model theory*. North Holland, Amsterdam, 1967.
23. J.-L. Lassez and M.J. Maher. On Fourier's algorithm for linear arithmetic constraints. *Journal of Automated Reasoning*, 9:373–379, 1992.
24. D. C. Luckham, S. M. German, F. W. Von Henke, R. A. Karp, P. W. Milne, D. C. Oppen, W. Polak, and W. L. Scherlis. Stanford Pascal Verifier user manual. Technical report, 1979. CSD Report STAN-CS-79-731, Stanford University, Stanford, CA.
25. Z. Manna. *STeP: the Stanford Temporal Prover*. Technical report, 1994. Technique Report STAN-CS-TR-94, Computer Science Department, Stanford.
26. A. Manning, A. Ireland, and A. Bundy. Increasing the Versatility of Heuristic Based Theorem Provers. In A. Voronkov, editor, *International Conference on Logic Programming and Automated Reasoning – LPAR 93, St. Petersburg*, number 698 in Lecture Notes in Artificial Intelligence, pages pp 194–204. Springer-Verlag, 1993.

27. G. Nelson and D. C. Oppen. Simplification by Cooperating Decision Procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, October 1979.
28. G. Nelson and D. C. Oppen. Fast Decision Procedures Based On Congruence Closure. *Journal of the ACM*, 27(2):356–364, April 1980. Also: Stanford CS Report STAN-CS-77-646, 1977.
29. S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 1996 Conference on Computer-Aided Verification*, number 1102 in LNCS, pages 411–414, New Brunswick, New Jersey, U. S. A., 1996. Springer-Verlag.
30. Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Sprawozdanie z I Kongresu metematyków slowiańskich, Warszawa 1929*, pages 92–101, 395. Warsaw, 1930. Annotated English version also available [33].
31. Harald Rueß and Natarajan Shankar. Deconstructing Shostak. 2000. draft version; available from http://www.csl.sri.com/shankar/shostak2000.ps.gz.
32. R. E. Shostak. Deciding Combinations of Theories. *Journal of the ACM*, 31(1):1–12, January 1984. Also: *Proceedings of the 6th International Conference on Automated Deduction*, volume 138 of *Lecture Notes in Computer Science*, pages 209–222. Springer-Verlag, June 1982.
33. Ryan Stansifer. Presburger's article on integer arithmetic: Remarks and translation. Technical Report TR 84-639, Department of Computer Science, Cornell University, September 1984.
34. Cesare Tinelli and Mehdi Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In Franz Baader and Klaus U. Schultz, editors, *Frontiers of combining systems: Proceeding of the 1st International workshop*, pages 103–120. Kluwer, 1996.
35. Hantao Zhang. Contextual Rewriting in Automated Reasoning. *Fundamenta Informaticae 24*, 24:107–123, 1995.